Introduzione a Processing

di MATTEO FIORAVANTI

Scopriamo come rappresentare un valore analogico acquisito con Arduino, realizzare interfacce grafiche e impariamo ad utilizzare la libreria "arduino". Terza puntata.

Abbiamo conosciuto Processing partendo dalle basi, scoprendo che cos'è e a quali ambiti si applica; dopo le prime due puntate, la prima delle quali propedeutica all'apprendimento dell'uso di questo ambiente di sviluppo per programmi da PC e la seconda dedicata all'interazione con Arduino ed allo sviluppo di programmi di test utilizzabili su quest'ultimo, siamo giunti alla terza lezione. Qui affronteremo tre temi fondamentali, ossia:

1) la rappresentazione di un valore analogico, letto sempre attraverso Arduino;
2) la costruzione di interfacce grafiche usando delle immagini e delle piccole animazioni;
3) l'utilizzo della libreria "arduino" di Processing, che ci consentirà di control-

lare direttamente i pin e le funzionalità di Arduino, senza passare per la definizione di un protocollo.

I programmi di esempio che vedremo in questa terza puntata del nostro corso sono elencati qui di seguito.

1. LEZIONE 3_1 PULSANTE + LED + INDICATORE ANALOGICO – Questo codice è la naturale prosecuzione della lezione della seconda puntata; in essa, all'interfaccia con un pulsante ed un LED viene aggiunta una barra che rappresenta il valore analogico letto da Arduino. Il codice che deve essere caricato su Arduino è:

`arduino_codice_TXRX_ANALOG.pde`.

2. LEZIONE 3_2 PULSANTE GRAFICO – Qui impariamo a costruire delle interfacce grafiche usando delle immagini; la tecnica adottata è molto simile a quella descritta nelle scorse lezioni, ma la differenza sostanziale è che non disegneremo da codice i pulsanti, i LED e gli indicatori, ma sfrutteremo delle immagini precedentemente costruite. Questo codice mostra come costruire un interruttore, che si evidenzia se ci portiamo sopra il puntatore del mouse e che commuta se vi facciamo clic.
3. LEZIONE 3_3 INDICATORE ANALOGICO GRAFICO – Questo codice serve a costruire un indicatore, simile ad un manometro, in cui c'è un'immagine di sfondo che rappresenta una scala graduata ed una lancetta che ruota. Tutto è realizzato mediante immagini precedentemente costruite e la lancetta ruota in funzione del valore che vogliamo visualizzare.
4. LEZIONE 3_4 MANOPOLA – In questo programma combiniamo le due tecniche delle lezioni precedenti per realizzare una manopola, che si evidenzia quando il mouse la punta e che se vi facciamo clic e ruotiamo il mouse, essa ruota conseguentemente.
5. LEZIONE 3_5 INTERFACCIA COMPLETA – In questo codice mettiamo insieme tutto quello che abbiamo appreso finora realizzando un'interfaccia completa di: interruttore, LED, manopola, indicatore analogico. L'interfaccia controllerà direttamente Arduino, nel quale dovrà essere caricato il codice:

`arduino_codice_TXRX_ANALOG_IN_OUT.pde`.

6. LEZIONE 3_6 INTERFACCIA COMPLETA + FIRMDATA – Utilizzando la stessa interfaccia grafica del codice precedente, andiamo ad usare la libreria arduino di Processing e carichiamo su Arduino il codice *firmdata*. Il programma mostra come controllare direttamente Arduino da Processing.

LEZIONE 3_1

Questo programma è la diretta evoluzione del codice spiegato nella seconda puntata nella LEZIONE 2_4. Richiamiamo brevemente cosa accadeva in quel programma: Arduino comunicava verso il programma master lo stato di un suo pulsante e faceva questo inviando un byte seriale in cui c'era il valore 1 o 0; nel caso fosse trasmesso il valore 1 il codice di Processing accendeva il secondo LED dell'interfaccia. Partendo da questa architettura, modifichiamo il codice e facciamo trasmettere ad Arduino non lo stato del pulsante, ma il valore analogico letto su ANALOG IN 0; dal lato Processing modifichiamo il codice in modo da decodificare il valore analogico ricevuto e rappresentarlo attraverso una barra. Il codice Arduino da utilizzare in questa lezione è:

`arduino_codice_TXRX_ANALOG.pde`,

un file che potete scaricare dal nostro sito *www.elettronicain.it* insieme a tutti i codici di esempio descritti. Ci soffermiamo, quindi, unicamente sulle parti di codice che ci interessano. Innanzitutto definiamo l'ingresso analogico 0, chiamandolo `inAnalogico`.

Nel ciclo di risposta alle interrogazioni del master, trasmettiamo il valore analogico letto sull'ingresso 0, quindi leggiamo il valore attraverso la funzione `analogRead()`, e lo dividiamo per 4, in quanto Arduino

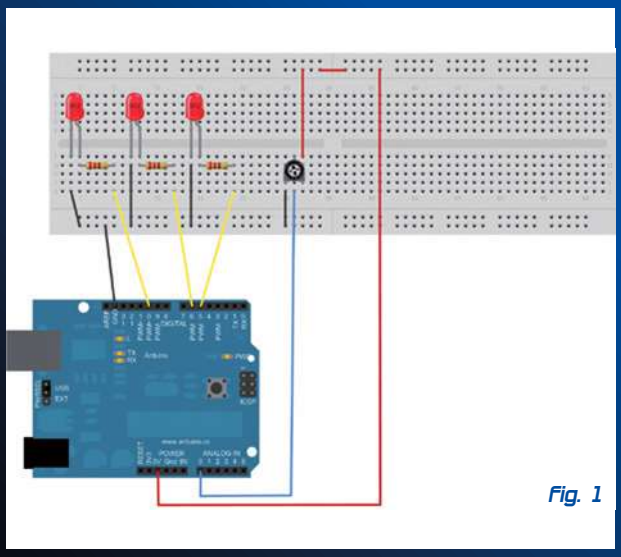

Fig. 1

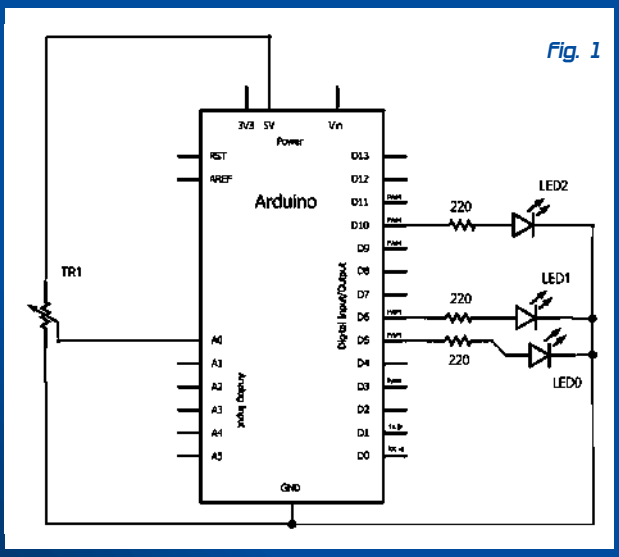

Fig. 1

dispone di convertitori analogico digitali a 10 bit e pertanto un'escursione della tensione d'ingresso tra 0 e 5 V corrisponde a una rappresentazione numerica intera 0–1024 (2^{10} = 1024). Siccome un byte è composto da 8 bit, il valore numerico massimo che possiamo trasmettere sulla seriale è 256 (2^8 = 256) ecco che dobbiamo comprimere la scala dividendo tutto per quattro.

```
anVal = analogRead(inAnalogico) / 4;
```

Il valore letto e scalato viene quindi inviato via seriale attraverso l'istruzione:

```
Serial.write(anVal);
```

Qualora avessimo voluto trasmettere il valore non scalato, ma rappresentativo di tutti e 10 i bit di conversione, avremmo dovuto spezzarlo trasmettendolo con due byte. Riportiamo nel Listato 1 le parti di codice di Arduino, mentre il circuito che utilizzeremo in questa lezione è rappresentato in Fig. 1; il relativo schema elettrico si trova nella Fig. 2. Dal punto di vista di Processing, il codice non differisce di molto da quello della LEZIONE 2_4; infatti le modifiche sono minime. Quando è presente un dato nel buffer seriale, questo è letto e caricato su *val*:

```
val = myPort.read();
```

La barra analogica è costruita mediante un rettangolo, di colore rosso, la cui dimensione "x" varia in funzione del valore di *val*:

```
rect(20,350,val,40);
```

La porzione di codice di Processing è riportata nel Listato 2; le Fig. 3 e Fig. 4, ci mostrano come si presenta l'interfaccia completa.
Occorre notare una cosa molto importante,

Listato 1

```
int inAnalogico = 0;
…

void loop() {

  //stato di lettura e scrittura della seriale

  if (Serial.available()) { // se è disponibile un dato sul buffer seriale
    digitalWrite(LEDPin[0], HIGH);  //accendi il LED 0, identificativo dello stato della comunicazione
    val = Serial.read(); //il valore del buffer è letto e salvato su val, una volta letto il buffer è svuotato

    anVal = analogRead(inAnalogico) / 4; // lettura dell'ingresso analogico e divisione per 4, (1024/4 = 256)
    Serial.write(anVal); // trasmissione del valore via seriale

  }
```

Listato 2

```
…

  if (myPort.available()>0) {   // se è presente un dato sul buffer seriale,
       val = myPort.read(); //il dato viene letto e memorizzato su val
       myPort.clear();      //il buffer è ripulito
      val = map(val, 0, 255, 0, 260); //il valore di ingresso varia 0-->255, la barra ha una dimensione 0-->260,
                           //la funzione map svolge automaticamente la proporzione
      }

  fill(0,0,0);         //colore di sfondo della barra, nero
  rect(20,350,260,40); //sfondo della barra
  fill(240,0,0);       //colore di riempimento della barra, rosso
  rect(20,350,val,40); //la barra analogica è un rettangolo la cui larghezza è proporzionale al valore analo-
gico rappresentato
  text("Analog", 20, 430);                        //etichetta

…
```

ossia che una volta letto, *val* subisce una trasformazione attraverso la funzione *map (value,low1,high1,low2,high2)*:

```
val = map(val, 0, 255, 0, 260);
```

in pratica *map()* trasforma il valore *value* (che deve essere compreso tra i valori *low1* e *high1*) in un corrispondente valore proporzionato che sarà compreso tra i valori *low2* e *high2*, come ci mostra la Fig. 5. Questa funzione è molto importante e sarà ampiamente usata nelle successive lezioni; in questo caso *val*, che è per sua natura compreso tra 0 e 255, viene impostato tra 0 e 260.
La motivazione di ciò è unicamente di tipo grafico, in quanto abbiamo definito la larghezza della barra analogica 260 pixel, mentre se avessimo definito una barra più larga (magari 500 pixel) avremmo dovuto impostare la funzione nel modo seguente:

```
val = map(val, 0, 255, 0, 500);
```

così facendo quando *val = 0*, la barra è tutta a 0, mentre quando *val = 255*, la barra occuperà tutti i 500 pixel.
Il funzionamento del sistema è riassunto in Fig. 6 e Fig. 7, il pulsante dell'interfaccia controlla il LED1 di Arduino, mentre LED0 si accende in ogni caso se è presente qualcosa nel buffer seriale. Il trimmer funziona da partitore di tensione ed il valore letto dall'ingresso analogico 0 di Arduino viene ritrasmesso via seriale verso l'interfaccia e qui rappresentato mediante la barra.

LEZIONE 3_2

Abbandoniamo per un po' l'interfacciamento e la comunicazione con Arduino, per concentrarci sulla costruzione di interfacce grafiche. L'obiettivo è realizzare pulsanti, LED, indicatori, attraverso delle immagini.
Nel mese scorso abbiamo costruito delle interfacce sfruttando le primitive grafiche di Processing, che sono essenzialmente quadrati, rettangoli, colori e riempimenti. Ora realizzeremo delle interfacce combinando quanto abbiamo finora appreso, con la possibilità di caricare immagini e sovrapporle.
Il codice di questa lezione si riferisce alla

Listato 3

```
…
void draw(){

  if (button == 0){  //se il valore di button = 0 viene visualizzata l'immagine del pulsante off
    image(imgBt1Off, 20,30);  //visualizzazione del pulsante off
  }
  else{                       //altrimenti
    image(imgBt1On, 20,30);   //viene visualizzata, nella stessa posizione, l'immagine del pulsaten on
  }

  text("button", 180, 130);                    //etichetta
…
```

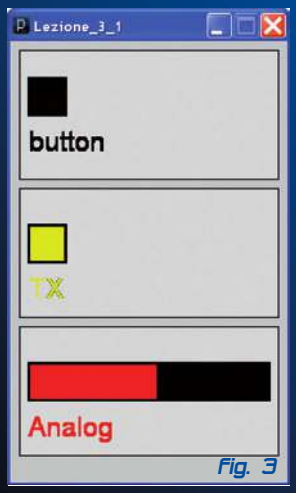

Fig. 3

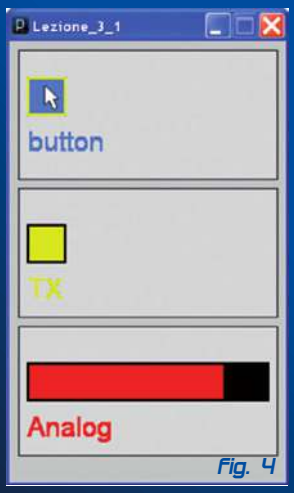

Fig. 4

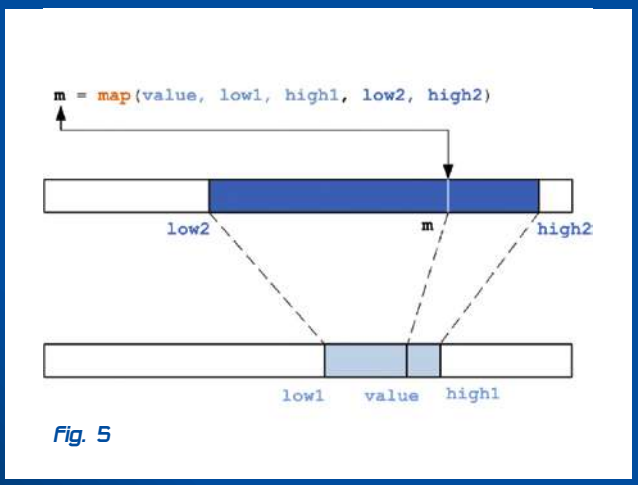

Fig. 5

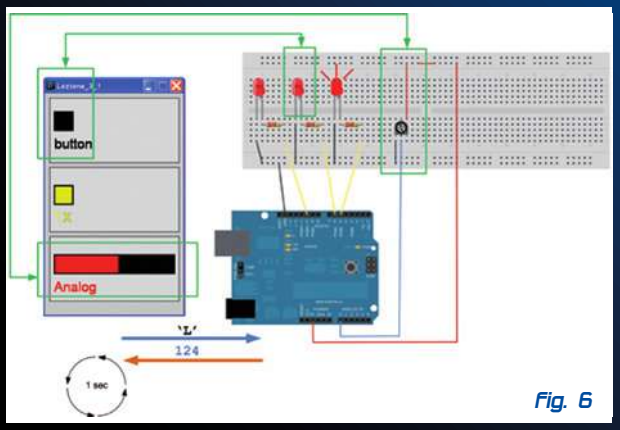

Fig. 6

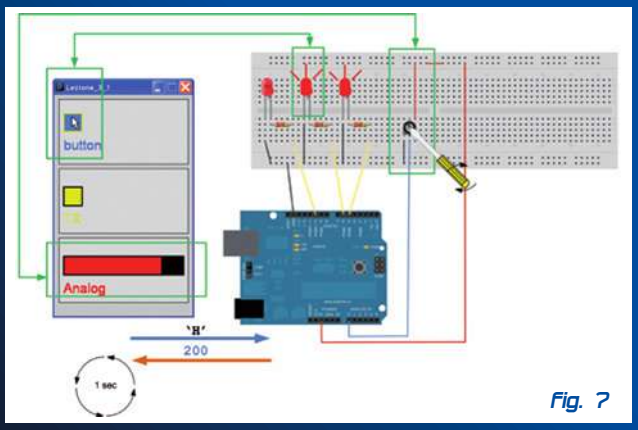

Fig. 7

costruzione di un pulsante e la Fig. 8 ci mostra come si presenta l'interfaccia; in essa c'è un interruttore con relativo LED. Quando il puntatore del mouse passa sopra l'interruttore, l'area corrispondente si evidenzia e quando facciamo clic l'interruttore si abbassa ed il LED si accende. Questa funzionalità è ottenuta semplicemente con due immagini: in una c'è rappresentato l'interruttore in posizione off con a fianco il LED spento, mentre nell'altra immagine si trova l'interruttore in posizione on con il LED acceso.
Il rettangolo che viene evidenziato è stato già descritto: sostanzialmente esso definisce l'area attiva in cui se facciamo clic cambieremo lo stato di una variabile e conseguentemente forzeremo il caricamento dell'immagine corrispondente (interruttore on oppure interruttore off).
Per gestire le immagini in Processing abbiamo bisogno delle seguenti primitive:

- PImage imgBt1On; questo è il tipo di dato per le immagini, in pratica *imgBt1On*, è una variabile definita del tipo immagine; Processing può gestire immagini .png, .gif, .jpg, .tga. In questo caso, oltre a definire le variabili tipo int, float, definiremo anche quelle Pimage;
- imgBt1On=loadImage("button2On.png"); questa istruzione, come si può facilmente intuire, costituisce il caricamento vero e proprio dell'immagine nella variabile *imgBt1On* che abbiamo precedentemente definito; l'immagine da caricare deve essere contenuta nella stessa cartella di Processing dove è contenuto il codice;
- image(imgBt1On, 0, 0); questa istruzione costituisce la visualizzazione dell'immagine (in questo caso nella posizione X0 e Y0); ovviamente, cambiando questi valori possiamo spostare l'immagine nell'area del programma.

Capito come si definiscono, caricano e visualizzano le immagini, la parte di codice del programma che riportiamo in questa puntata del corso è quella visibile nel Listato 3.
A cambiare il valore della variabile *button* sarà il fatto che avremo fatto clic nell'area attiva; questo evento sarà gestito dalla

Fig. 8

funzione overRect(). Ciò che è interessante notare è che, ad ogni ciclo, possiamo rappresentare un'immagine o l'altra e che l'effetto dell'interruttore abbassato e del LED acceso è dato semplicemente dalla sovrapposizione di due immagini.
La costruzione delle immagini grafiche per le interfacce può essere fatta con qualsiasi programma di disegno: quasi banalmente con paint, oppure con Inkscape; addirittura potremmo caricare delle foto o combinare foto e disegni.

LEZIONE 3_3

Estendiamo ora l'applicazione delle immagini per realizzare un indicatore analogico, tipo manometro, rappresentato nella Fig. 9. L'indicatore è ottenuto per mezzo della sovrapposizione di due immagini, una delle quali rappresenta la scala graduata e sarà tenuta fissa, mentre l'altra rappresenta la lancetta e sarà fatta ruotare. Le immagini che ci servono sono *gauge.png* (la scala) e *arrow.png* (la lancetta). In queste applicazioni è conveniente usare le immagini .png, in quanto, pur essendo del tutto analoghe alle bitmap, hanno il vantaggio di poter definire il colore trasparente, come evidenziato nella Fig. 10. In questa stessa figura vediamo che le nostre immagini hanno dimensioni di 300x300 pixel per la base e 120x120 pixel per la lancetta.
Nel programma definiremo le variabili immagine, che sono le seguenti:

```
PImage gauge;
PImage arrow;
```

Poi caricheremo le immagini effettive::

```
gauge = loadImage("gauge2.png");
arrow = loadImage("arrow2.png")
```

A questo punto vediamo che se andassimo a visualizzarle in questo modo:

```
image(gauge, 0, 0);
image(arrow, 0, 0);
```

otterremmo quanto rappresentato nella Fig. 11. Tuttavia vogliamo assicurarci che i centri della scala e della lancetta coincidano e che la lancetta possa essere fatta ruotare.
Per ottenere queste funzionalità ci servono ancora due primitive molto importanti di Processing, che sono translate() e rotate(); la prima effettua una traslazione, mentre la seconda opera una rotazione. Dobbiamo però prestare molta attenzione al fatto che non si riferiscono alle immagini, ma al sistema di riferimento.
In Processing, ciò che viene traslato e ruotato è sempre il sistema di riferimento.
La Fig. 12 illustra come avviene questo processo, che possiamo spiegare così: supponiamo di voler porre l'immagine della lancetta al centro e ruotata di 30°; innanzitutto trasliamo il sistema di riferimento alla coordinata 150,150. Questo punto ora sarà la nuova origine 0,0. Ora, se ruotiamo tutto di 30° ruoterà l'intero sistema di riferimento. Se rappresentassimo

Fig. 9

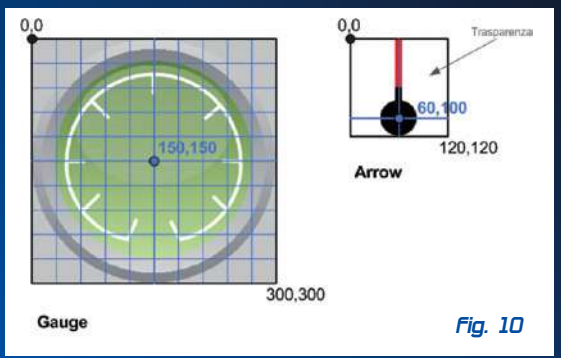

Fig. 10

Listato 4

```
PImage gauge; //definizione dell'immagine della base dell'indicatore
PImage arrow; //definizione dell'immagine della lancetta

int val;                 //definizione della variabile che definisce la rotazione e che varierà tra 0 - 255
float valm;             //definizione della variabile che definisce la rotazione e che varierà tra -160 e +160

void setup(){
  color(RGB);            //tipo di colore
  strokeWeight(1);       //spessore linee
  strokeJoin(SQUARE);    //tipo di spigolo: quadrato

  fill(30);              //grigio di sfondo
  smooth();
  size(300,300);

  gauge = loadImage("gauge2.png"); //caricamento dell'immagine relativa alla base dell'indicatore
  arrow = loadImage("arrow2.png"); //caricamento dell'immagine relativa alla lancetta
}

void draw(){
  image(gauge,0,0);    //visualizzazione dell'immagine della base dell'indicatore
  translate(150,150);  //traslazione dell'origine del sistema di riferimento da X0,Y0 a X150,Y150

  val=250;       //attribuiamo a val un valore fisso, val 0-->255
  valm = map(val, 0, 255, -160, 160); //valm è la traduzione di val in un range che va da -160 a +160
                        //quando val = 0 valm = -160, quando val = 255 valm = +160

  rotate(radians(valm)); //rotazione del sistema di riferimento, il valore di valm (espresso in °) viene tra-
dotto in radianti
  image(arrow,-60,-100); //nel sistema di riferimento ruotato e traslato viene rappresentata l'immagine della
lancetta


}
```

ora l'immagine della lancetta:

```
image(arrow, 0, 0)
```

otterremmo quanto rappresentato dalla figura centrale e cioè la lancetta ruotata di 30°. Ma rispetto alla sua origine (0,0;) per centrare la lancetta rispetto al centro della scala dobbiamo posizionarla al punto definito da x=-60 e y=-100. Se faremo in modo che l'angolo di rotazione sia rappresentativo del valore da visualizzare, avremo completato il nostro indicatore. Riportiamo, nel Listato 4, il relativo codice di Processing. Cambiando il valore di *val*, tra 0 e 255, si nota che la lancetta percorre tutta la rotazione dal valore iniziale della scala a quello finale.

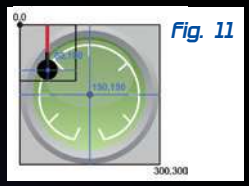

Fig. 11

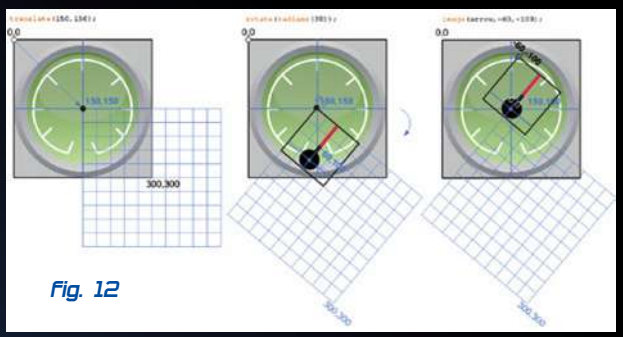

Fig. 12

Ad agire sulla rotazione della lancetta non è direttamente *val*, ma *valm*, cioè la traduzione di *val* che va da 0 a 255 in un valore che invece varia tra -160 e +160. Perciò quando *val* è 0, allora valm=-160, mentre quando val=128 → valm = 0 (lancetta verticale); ancora, quando val = 255 → valm = 160 (fondo scala).
Questa conversione è ottenuta per mezzo della funzione map() precedentemente descritta.
La variabile *valm* rappresenta un valore espresso in °, ma invece la funzione rotate() agisce su un valore angolare espresso in radianti; il problema viene risolto semplicemente attraverso la funzione di conversione radians().

LEZIONE 3_4

Avendo definito un pulsante ed un indicatore grafico, realizziamo ora un altro strumento di input, cioè una manopola. Si tratta sostanzialmente di un controller rotativo funzionante così: quando il punta-

Listato 5

```
…

if ((overRect(10,10,100,100)) && mousePressed){  //se contemporaneamente il mouse è sopra l'area della manopola
ed il tasto sinistro è premuto
                          knobRot = int(map(mouseX,10, 110, -130,130)); //rotazione, mouseX varia tra 10 e
110, map fa la proporzione tra -130° e +130°
                           valTx = int(map(mouseX,10,110,0,255)); //valTx potrebbe essere valore trasmesso
via seriale, mouseX varia tra 10 e 110, viene scalato tra 0 e 255
                          println(valTx); //trasmissione a terminale di valTx
  }


  rotate(radians(knobRot));  //rotazione del sistema di riferimento in base al valore della variabile knobRot
  image(knob, -40,-40);      //visualizzazione della manopola ruotata
...
```

tore del mouse gli passa sopra, si evidenzia, mentre se vi facciamo clic muovendo contemporaneamente il mouse, la manopola ruota. La funzionalità corrispondente è illustrata nella Fig. 13. La tecnica che fa evidenziare il controller è sempre la stessa, ovvero usiamo la funzione overRect(). Per quanto riguarda la rotazione della manopola, invece, il principio è analogo alla rotazione dell'immagine della lancetta nell'indicatore analogico, ma l'angolo di rotazione deve dipendere dallo spostamento del mouse.
Riportiamo, nel Listato 5, la porzione di codice che permette di realizzare le funzionalità descritte. Se il mouse è sopra l'area attiva ed il tasto sinistro è premuto (mousePressed) allora si verifica:

Fig. 13

```
knobRot = int(map(mouseX,10, 110, -130,130))
```

Alla variabile *knobRot*, viene assegnato un valore che va da -130 a +130 e questo valore è proporzionato rispetto a 10÷100, che è l'escursione del mouse nell'area attiva (la posizione "x" del puntatore del mouse viene restituita dal parametro *mouseX*). Quando il cursore del mouse si trova nell'area attiva, definita nell'asse x da 10 a 100, ed il tasto sinistro è premuto, muovendo il mouse il parametro *mouseX* varierà tra 10 e 100.
La variabile *knobRot* rappresenterà un angolo di rotazione che varierà e tra -130° e +130°. Questo angolo è rappresentativo di *mouseX* (che, come abbiamo detto, varia tra 10 e 100), ma varierà nel'ambito compreso fra -130 e +130.

LEZIONE 3_5

Sfruttando tutte le tecniche di visualizzazione, di rappresentazione e controllo fin qui descritte, è possibile

Listato 6

```
…

  //stato di decodifica del valore seriale ricevuto

  if (val == 'H') { // se il valore di val è il carattere 'H'
    digitalWrite(LEDPin[1], HIGH); // viene acceso il LED 1
  }
  if (val == 'L'){
    digitalWrite(LEDPin[1], LOW);
  }

  if ((val >0) && (val<255)){ // se il valore seriale ricevuto è compreso tra 0 e 255
    analogWrite(LEDPin[2], val); // sul terzo pin della lista, quindi pin 10 viene rappresentato val in PWM
  }


  delay(1); // ciclo di attesa di 1 ms
…
```

Fig. 14

Fig. 15

realizzare un'interfaccia utente completa. L'interfaccia, riportata in Fig. 14 e Fig. 15, è sostanzialmente la stessa descritta nella seconda puntata, ma in questo caso il pulsante è stato sostituito da due immagini grafiche e lo stesso vale per il LED; inoltre sono stati aggiunti la manopola e l'indicatore analogico grafico. Il pulsante controlla il LED 1 di Arduino, mentre la posizione della manopola controlla il PWM che Arduino genererà sul LED 2; l'indicatore analogico rappresenta il valore letto da Arduino su Analog In 0, mentre il LED che si accende ogni secondo rappresenta il momento in cui il programma comunica con Arduino.
Il codice Processing dell'interfaccia è dunque la somma dei codici qui descritti, solamente che le posizioni geometriche dei vari elementi e delle aree attive sono diverse.
La Fig. 16 mostra il sistema al completo.
Il codice per Arduino che deve essere utilizzato in questa lezione pratica è contenuto nel file *arduino_codice_TXRX_ANALOG_IN_OUT.pde*.
Questo differisce dal precedente codice soltanto per il fatto che ha in più la decodifica del valore PWM da rappresentare sull'uscita. Il Listato 6 contiene il codice per Arduino.

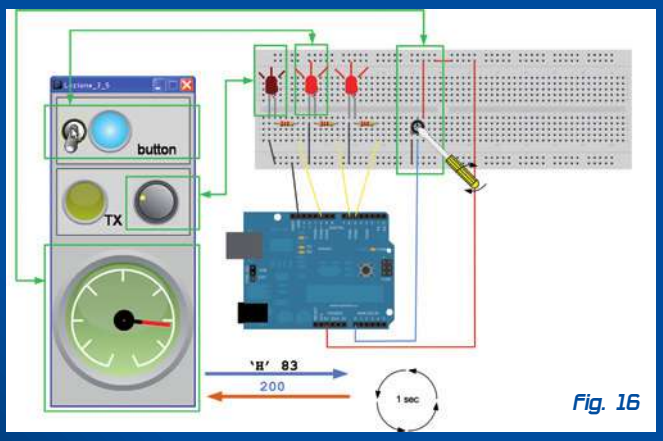

Fig. 16

LEZIONE 3_6

Rivediamo ora lo stesso programma della Lezione 3_5, utilizzando però una nuova tecnica di controllo di Arduino: in pratica sfruttiamo un'apposita libreria di Processing che ci offre la comodissima opportunità di controllare direttamente i pin di Arduino da Processing, come se Arduino fosse una naturale estensione del PC. Così realizziamo un'interfaccia completa con *firmdata*.
Ovviamente non basta soltanto richiamare la libreria, perché dovremo anche far girare su Arduino un apposito programma chiamato *firmdata*.
Ma procediamo con ordine: per prima cosa visitiamo la pagina web *www.arduino.cc/playground/Interfacing/Processing*, dalla quale scarichiamo la libreria *processing-arduino-0017.zip*; questa deve essere estratta e la cartella *arduino* va copiata nella cartella *libraries* di Processing.
A questo punto bisogna aprire l'editor di Arduino e, come mostrato dalla Fig. 17, accedere al percorso *Examples>Firmdata >StandardFirmadata*, quindi fare *verify* e l'upload su Arduino.
A questo punto i programmi in Processing dovranno essere strutturati considerando che innanzitutto bisognerà importare la libreria *arduino* e la libreria seriale:

```
import processing.serial.*;
import cc.arduino.*;
```

Poi occorrerà definire l'oggetto *arduino* della classe Arduino (questi aspetti saranno meglio approfonditi nelle prossime lezioni):

```
Arduino arduino;
```

Ora, dentro Setup definiremo le caratteristiche di comunicazione di Arduino, analogamente a quanto si faceva per la porta seriale:

```
void setup()
{
  arduino = new Arduino(this, Arduino.list()[0], 57600);
}
```

Descriviamo, qui di seguito, le primitive

Listato 7

```
translate(150,480);
val=arduino.analogRead(0);    //il dato viene letto e memorizzato su val
valm = map(val, 0, 1024, -160, 160); //valm è la traduzione di val in un range che va da -160 a +160
                                     //quando val = 0 valm = -160, quando val = 1024 valm = +160
rotate(radians(valm));  //rotazione del sistema di riferimento, il valore di valm (espresso in °) viene tra-
dotto in radianti

image(arrow,-60,-100);  //nel sistema di riferiemento ruotato e traslato viene rappresentata l'immagine della
lancetta
```

che abbiamo a disposizione usando la libreria arduino.

pinMode(pin, mode): setta un pin digitale come ingresso o come uscita (*Arduino.INPUT* o *Arduino.OUTPUT*).
Esempio:

```
arduino.pinMode(6, Arduino.OUTPUT); il pin 6 di Arduino è settato come uscita
arduino.pinMode(7, Arduino.INPUT); il pin 7 di Arduino è settato come ingresso
```

digitalRead(pin): legge lo stato di un pin digitale, precedentemente settato come ingresso, e restituisce il valore (*Arduino.LOW* o *Arduino.HIGH*).
Esempio:

```
arduino.digitalRead(7); legge lo stato del pin 7
```

digitalWrite(pin, value): scrive *Arduino.LOW* o *Arduino.HIGH* su di un pin digitale settato come uscita.
Esempio:

```
arduino.digitalWrite(5, Arduino.LOW); setta come alto il pin 5
arduino.digitalWrite(5, Arduino.HIGH); setta come basso il pin 5
```

analogRead(pin): restituisce il valore di un ingresso analogico (0 – 1023), i pin analogici di Arduino sono: 0,1,2,3,4.
Esempio:

```
val=arduino.analogRead(0); legge il valore dell'ingresso analogico 0
```

analogWrite(pin, value): genera un segnale PWM sul pin assegnato. I pin di Arduino che supportano la generazione del segnale PWM sono: 3, 5, 6, 9, 10, 11. Il campo value varia tra 0 (PWM 0%) e 255 (PWM 100%).
Esempio:

```
arduino.analogWrite(10, valTx); scrive sul pin 10 un valore PWM che è dato dalla variabile valTx
```

Nel Listato 7 riportiamo la porzione di codice relativa alla visualizzazione della condizione dell'ingresso analogico. Vediamo che, in pratica, per leggere il valore basta semplicemente richiamare l'opportuna funzione ed indirizzare la lettura nel pin di Arduino che ci interessa. Questa tecnica è estremamente comoda e versatile, ma ha lo svantaggio che Arduino deve essere programmato con *firmdata*. Se vogliamo far svolgere ad Arduino delle funzioni in maniera autonoma, è necessario prevedere un'architettura master/slave, nella quale Arduino svolge le sue operazioni e quando interpellato dal master risponde; per questa ragione Arduino sarà programmato con un codice che prevede, oltre alle operazioni da svolgere, anche il protocollo di comunicazione con il master. Se invece vogliamo considerare Arduino come una periferica e tutto il programma risiede nel PC, allora conviene sfruttare la tecnica qui descritta.

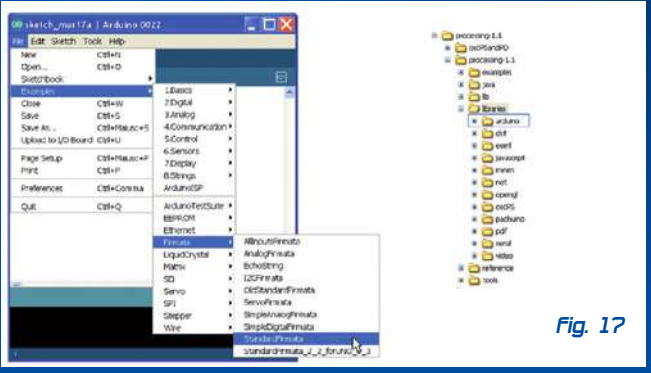

Fig. 17

PROSSIME LEZIONI

Nella prossima puntata introdurremo la programmazione ad oggetti, che ci permetterà di costruire gli elementi dell'interfaccia quali pulsanti, LED, indicatori ecc., in una maniera molto più versatile e ci consentirà di costruire interfacce complesse in modo abbastanza semplice.